UNIVERSIDADE FEDERAL DE VIÇOSA
Campus Rio Paranaíba
Sistemas de Informação
Inteligência Artificial - SIN 323

RELATÓRIO: PROJETO 01

6954 - Victor Alves de Oliveira

12/2024

1. Sobre o Projeto

Este projeto tem como objetivo implementar e analisar algoritmos de busca aplicados à resolução de labirintos em um ambiente determinístico e totalmente observável. A proposta central é desenvolver um agente, que utilizando diferentes estratégias de busca, consiga navegar eficientemente pelo labirinto, evitando ciclos e alcançando o objetivo no menor número de passos possível.

A implementação é modular, garantindo que sensores, atuadores e características do ambiente possam ser facilmente configurados ou substituídos via código fonte. Isso inclui alterações no tamanho do labirinto, nos algoritmos de busca utilizados e nas condições do ambiente. O agente utiliza algoritmos clássicos como Busca em Largura (BFS) e Busca em Profundidade (DFS), além de uma abordagem heurística (Busca em Feixe - BS), proporcionando uma análise comparativa de suas eficácias no contexto proposto.

Os parâmetros principais para a formulação do problema incluem:

- **Estado Inicial:** A posição inicial do agente no labirinto.
- **Sucessores:** Movimentos permitidos para as células adjacentes (cima, esquerda, direita, baixo), respeitando as regras de navegação.
- **Teste de Objetivo:** A verificação de que o agente alcançou a célula-alvo.
- **Custo do Caminho:** A métrica usada para avaliar o desempenho do agente, calculada com base no número de passos necessários para completar o labirinto.

O projeto também explora elementos como:

- **Interface Gráfica:** Para visualizar os movimentos do agente em tempo real, contribuindo para uma melhor compreensão dos algoritmos.
- **Avaliação Comparativa:** Comparação de desempenho entre diferentes estratégias de busca, considerando eficiência e precisão.

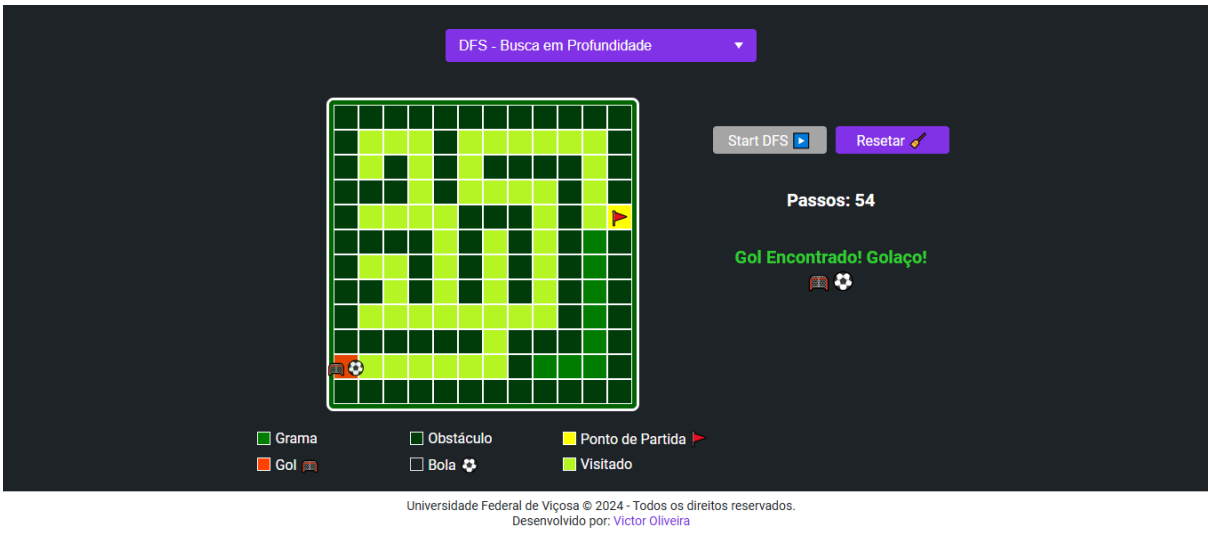
Além disso, o ambiente é determinístico, o que significa que cada ação tem um resultado previsível, e o agente possui percepções completas do ambiente, incluindo a localização atual e as informações sobre células vazias ou bloqueadas.

2. Tecnologias Utilizadas

- Visual Studio Code = IDE e Editor de Código
- Git = Versionamento de Código
- Github = Repositório do Projeto
- React Js v18.3.1 = Biblioteca Gráfica
- Javascript = Linguagem de Programação
- CSS v3: Estilização do projeto
- Vercel = Plataforma de hospedagem para websites

3. Como Executar o Projeto

Consulte a execução local do projeto atualizada no arquivo Readme.md no repositório: https://github.com/VictorAlves08/agent_maze_navigator
ou
Acesse a aplicação na web: https://agent-maze-navigator.vercel



4. Funcionalidades

1 - Seleção de Algoritmos de Busca:

- Permite ao usuário escolher entre diferentes algoritmos de busca:
  - Busca em Largura (BFS)
  - Busca em Profundidade (DFS)
  - Busca em Feixe (BS)

2 - Visualização em Tempo Real:

- Mostra a movimentação do agente no labirinto, destacando as células visitadas.

- Diferencia visualmente os estados inicial, final, e as paredes do labirinto.
- Exibe em tempo real o número de passos dados pelo agente durante a execução do algoritmo.
- Indica claramente quando o objetivo foi alcançado.
- Destaque animado para o agente (⚽) e o objetivo (🥅).
- Explicações visuais na interface sobre o significado das diferentes células do labirinto.

3 - Interface Responsiva:

- Compatível com dispositivos móveis, tablets e desktops.

5. Arquitetura de pastas do Projeto

A estrutura facilita a aplicação de estilos responsivos e boas práticas de desenvolvimento.

Pasta public/

- Contém arquivos estáticos de configuração do React Js.

Pasta src/

- **Pasta algorithms/:** Implementações dos algoritmos de busca de forma modular. Cada algoritmo é isolado, permitindo fácil substituição ou adição de novos.
- **Pasta components/:** Componentes React organizados por funcionalidade:
  - **Pasta Footer/:** Componente responsável pelo rodapé, com informações de créditos e licença.
  - **Pasta Maze/:** Componente principal que renderiza o labirinto, controla o agente e exibe passos.
  - **Pasta SelectAlgorithm/:** Dropdown interativo para selecionar o algoritmo de busca.
- **Pasta contexts/:** Context API para gerenciar o estado global, como o algoritmo selecionado e o status da execução.
- **Pasta utils/:** Contém constantes e dados essenciais para o labirinto e opções de configuração.

Arquivo index.css

- Define os estilos globais da aplicação, como cores, fontes e responsividade.

Arquivo App.js

- Componente raíz que organiza os principais componentes (SelectAlgorithm, Maze e Footer).

Arquivo package.json

- Lista as dependências do projeto, como React, scripts para iniciar, testar ou construir a aplicação.

6. Explicação das Algoritmos

O projeto implementa três algoritmos clássicos de busca aplicados à resolução de labirintos. Cada algoritmo utiliza uma abordagem distinta para explorar o espaço de busca, sendo analisados em termos de eficiência, memória e caminho encontrado.

1 - Busca em Profundidade (DFS):

- Como Funciona: Explora profundamente cada caminho antes de retroceder para explorar alternativas. Usa uma pilha (LIFO) para gerenciar os estados visitados.
- **Vantagens:**
  - Consome menos memória.
  - Fácil de implementar.
- **Desvantagens:**
  - Não garante o caminho mais curto.
  - Pode explorar caminhos longos e ineficazes antes de encontrar a solução.
- **Aplicação Ideal:** Ambientes onde o objetivo está localizado em profundidades maiores e a memória disponível é limitada.

2 - Busca em Largura (BFS):

- **Como Funciona:** Explora todos os nós em um nível antes de passar para o próximo. Usa uma fila (FIFO) para gerenciar os estados visitados.
- **Vantagens:**
  - Garante o menor caminho em termos de número de arestas.
  - Ideal para encontrar soluções eficientes em termos de distância.
- **Desvantagens:**
  - Alto consumo de memória, especialmente em grandes espaços de busca.
- Aplicação Ideal: Ambientes onde o objetivo está próximo ao ponto inicial e a eficiência do caminho é crucial.

3 - Busca em Feixe (BS):

- **Como Funciona:** Uma variação heurística da BFS que limita o número de estados explorados por nível com base em uma largura de feixe (beam width). Usa uma heurística (distância de Manhattan) para priorizar os estados mais promissores.

- **Vantagens:**
  - Mais eficiente em termos de memória.
  - Balança entre exploração e eficiência.
- **Desvantagens:**
  - Não garante o caminho mais curto, pois pode descartar soluções viáveis devido à limitação do feixe.
- **Aplicação Ideal:** Cenários onde a memória disponível é restrita e soluções rápidas são preferidas.

7. Análise e Resultados Obtidos

Após a execução dos algoritmos no labirinto proposto, os seguintes resultados foram registrados:

1 - Busca em Feixe (BS):

- **Resultado:** Obteve o menor número de **passos (50)**, indicando um caminho eficiente, embora não garantido como o mais curto.
- **Eficiência:** Beneficiou-se da heurística utilizada, priorizando caminhos promissores e descartando estados menos relevantes.

2 - Busca em Profundidade (DFS):

- **Resultado:** Completou o labirinto em **54 passos**, mostrando um desempenho intermediário.
- **Comportamento:** Explorou caminhos profundamente antes de retroceder, conseguindo evitar ciclos e alcançar o objetivo de forma eficiente.

3 - Busca em Largura (BFS):

- **Resultado:** Necessitou de **56 passos**, sendo o menos eficiente neste cenário.
- **Comportamento:** Explorou todos os nós em cada nível, o que resultou em passos adicionais antes de alcançar o objetivo.

Conclusões

- **BS se destacou pela eficiência em passos**, confirmando a eficácia da heurística combinada com a limitação de estados explorados.
- **DFS mostrou-se competitivo,** especialmente em um labirinto com múltiplos caminhos e obstáculos, indicando que sua abordagem direta pode ser eficiente em determinados cenários.
- **BFS, embora teoricamente mais preciso em encontrar o menor caminho**, foi o menos eficiente neste caso específico devido ao alto número de estados explorados.